

RemarksI. Background

In an Advisory Action dated April 3, 2006, the Examiner stated:

Applicant maintains that the prior art (Bennett) is not enabled even after the examiner pointed out in the previous Advisory Action that there is a retransmission mechanism preventing Bennett's network card from acknowledging lost packets. To revisit the issue, Applicant is directed to lines 3-5 at page 4 of the recent remarks quoting Comer's article: "The sender keeps a record of each packet it sends and waits for an acknowledgment before sending the next packet...". Thus, as long as the ACK packet is sent out in response to valid packet been received (see, e.g., Bennett: col. 12, lines 7-11), the above hand-shaking type of protocol would not yield any lost packet been incorrectly acknowledged.

Applicant is reminded that the above opinion had been conveyed to Applicant's representative, Mr. Lauer, on March 17, 2006 over a telephone interview, during which the examiner also offered suggestions regarding possible ways to overcome the prior art of record. As such, it is believed that this enablement issue has been clearly clarified.

II. Applicants' Response

Applicants' previous arguments demonstrating the deficiency of the Final Rejection and the lack of enablement of Bennett will not be reiterated here. The Examiner's statements regarding the telephone interview between the Examiner and the undersigned, and his rationale for claiming that Bennett is enabled, however, merit comment.

The Examiner did offer a suggestion in the interview for amending the claims to a form that he stated would be patentable, but the undersigned respectfully declined making such an amendment because the Final Rejection is not supported.

The Examiner then explained why he thought that Bennett was enabled. The Examiner stated that if one sent a single packet and then waited for an acknowledgment to arrive for that single packet or a retransmission timeout to occur for that single packet before sending the next single packet that Bennett could be made to work without losing packets.

The undersigned responded that there is no teaching in Bennett of sending a single packet and then waiting for an acknowledgment to arrive for that single packet or a retransmission timeout to occur for that single packet before sending the next single

packet, and so the 102 rejection is still incorrect. The undersigned also noted that such a modification of Bennett would not be obvious because it would require the other side of a TCP connection to send a single packet at a time so that the side that contained Bennett's device could perform without error, and because it would slow TCP communication to a crawl. The undersigned also notes that the Examiner's proposal contradicts Bennett's objective of "efficiently operating a reliable communication protocol," as well as eviscerating basic functions of TCP, such as the sliding window protocol.

Applicants further respectfully submit that the Examiner's quotation from Comer that "The sender keeps a record of each packet it sends and waits for an acknowledgment before sending the next packet..." is taken out of context. Applicants provided the pages of Comer because in the Advisory Action of February 28, 2006, the Examiner did not seem to understand the retransmission mechanism of TCP, and that quote offered a basic building block for understanding retransmission for "*most reliable protocols*."¹ TCP, however, requires a sliding window protocol that provides a mechanism to communicate multiple packets between acknowledgments without overloading the receiver. Comer begins discussing this basic function of TCP later, on page 175, by stating:

12.5 The Idea Behind Sliding Windows

Before examining the TCP stream service, we need to explore an additional concept that underlies stream transmission. The concept, known as a *sliding window*, makes stream transmission efficient...

*A simple positive acknowledgement protocol wastes a substantial amount of network bandwidth because it must delay sending a new packet until it receives an acknowledgement for the previous packet.*²

The fact that TCP uses sliding windows and sends out multiple packets before receiving an acknowledgement for the first packet is well known to those of ordinary skill in the art and discussed, for example, on page 187 of Comer, which states:

The TCP acknowledgement scheme is called *cumulative* because it reports how much of the stream has accumulated. Cumulative acknowledgements have both advantages and disadvantages. One advantage is that acknowledgements are both easy to generate and unambiguous. Another advantage is that lost acknowledgements do not

¹ Douglas E. Comer, "Internetworking with TCP/IP," Volume 1, (1991), page 173, line 38, emphasis added. A copy of pages 173-175 and 187 of Comer was provided with the Second Request for Reconsideration.

² Comer, page 175, lines 19-32, emphasis in original.

necessarily force retransmission. A major disadvantage is that the sender does not receive information about all successful transmissions, but only about a single position in the stream that has been received.³

On the same page, Comer states:

Because (TCP) segments travel in IP datagrams, they can be lost or delivered out of order; the receiver uses the sequence number to reorder segments.⁴

This statement also makes clear that TCP does not wait to receive an acknowledgement for each packet before sending the next packet, because the packets could in that case never arrive out of order. Even Bennett recognizes that TCP segments can be received out of order,⁵ although it fails to recognize that its automatic ACK generation would destroy the basic functions and reliability of TCP.

Stevens also notes that TCP uses a sliding window protocol to send multiple packets before waiting for an acknowledgement, as is well known to those of ordinary skill in the art, distinguishing the mechanism proposed by the Examiner by stating:

In Chapter 15 we saw that TFTP uses a stop-and-wait protocol. The sender of a data block required an acknowledgement for that block before the next block was sent. In this chapter we'll see that TCP uses a different form of flow control called a *sliding window* protocol. It allows the sender to transmit multiple packets before it stops and waits for an acknowledgement. This leads to faster data transfer, because the sender does not have to stop and wait for an acknowledgement each time a packet is sent.⁶

Applicants respectfully submit that TCP is so complicated that neither Bennett nor the Examiner understood some of its most basic functions, indicating the nonobviousness of the pending claims. Applicants again respectfully request the Examiner to reconsider the Final Rejection, because the Final Rejection has not presented a *prima facie* case of anticipation or obviousness for any of the claims, and because the primary reference relied upon for that rejection is nonenabled. As such, applicants

³ Comer, page 187, lines 18-24, italics in original, underline added.

⁴ Comer, page 187, lines 6-8.

⁵ Bennett, column 11, line 66 – column 12, line 2.

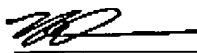
⁶ Richard Stevens, "TCP/IP Illustrated, Volume 1, The Protocols" (1994), page 275, lines 3-8, emphasis in original. For the Examiner's convenience, a copy of page 275 of Stevens is enclosed.

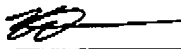
respectfully assert that the Final Rejection should be withdrawn, and a Notice of Allowance provided.

Respectfully submitted,

CERTIFICATE OF FACSIMILE TRANSMISSION
I hereby certify that this correspondence is being transmitted
via facsimile to the Commissioner for Patents, P.O. Box 1450,
Alexandria, VA 22313, telephone number (571) 273-8300, on
April 11, 2006.

Date: 4-11-06


Mark Lauer


Mark Lauer
Reg. No. 36,578
6601 Koll Center Parkway
Suite 245
Pleasanton, CA 94566
Tel: (925) 484-9295
Fax: (925) 484-9291

UNIX is a technology trademark of X/Open Company, Ltd.

The publisher offers discounts on this book when ordered in quantity for special sales.
For more information please contact:

Corporate & Professional Publishing Group
Addison-Wesley Publishing Company
One Jacob Way
Reading, Massachusetts 01867

Library of Congress Cataloging-in-Publication Data
Stevens, W. Richard

TCP/IP Illustrated: the protocols/W. Richard Stevens.
p. cm. — (Addison-Wesley professional computing series)
Includes bibliographical references and index.
ISBN 0-201-63346-9 (v. 1)
1. TCP/IP (Computer network protocol) I. Title. II. Series.

TK5105.55S74 1994

004.6'2—dc20

Copyright © 1994 Addison Wesley Longman, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission of the publisher. Printed in the United States of America. Published simultaneously in Canada.

Text printed on recycled and acid-free paper.

ISBN 0-201-63346-9

13 1415161718 MA 02 01 00 99

13th Printing January 1999

20

TCP Bulk Data Flow

20.1 Introduction

In Chapter 15 we saw that TFTP uses a stop-and-wait protocol. The sender of a data block required an acknowledgment for that block before the next block was sent. In this chapter we'll see that TCP uses a different form of flow control called a *sliding window* protocol. It allows the sender to transmit multiple packets before it stops and waits for an acknowledgment. This leads to faster data transfer, since the sender doesn't have to stop and wait for an acknowledgment each time a packet is sent.

We also look at TCP's PUSH flag, something we've seen in many of the previous examples. We also look at slow start, the technique used by TCP for getting the flow of data established on a connection, and then we examine bulk data throughput.

20.2 Normal Data Flow

Let's start with a one-way transfer of 8192 bytes from the host svr4 to the host bsd1. We run our sock program on bsd1 as the server:

```
bsd1 % sock -i -s 7777
```

The -i and -s flags tell the program to run as a "sink" server (read from the network and discard the data), and the server's port number is specified as 7777. The corresponding client is then run as:

```
svr4 % sock -i -n8 bsd1 7777
```

This causes the client to perform eight 1024-byte writes to the network. Figure 20.1 shows the time line for this exchange. We have left the first three segments in the output to show the MSS values for each end.